1 Dates

- Assignment 2 handin due: Friday, February 13, 2009
- To hand in, run /course/cs148/bin/cs148_handin asgn2_seeking

2 Introduction

In Assignment 1, you were introduced to the Player/Stage/Gazebo (PSG) robot interface and simulation system. This assignment demonstrated how to interact with real SmURV robots using the Player robot middleware. You should now be familiar with reading sensor data and sending control commands via manual controls (playerv) and/or a client program (client.cpp).

In Assignment 2, you will be extending your control client to perform an object seeking task. In this seeking task, your robot will look for and drive to objects (and "fiducials" or "landmarks") that are recognizable from visual sensing (i.e., camera). For this assignment, you will be working primarily with the SmURV platform with a Unibrain Fire-i video camera. The Player camera proxy provides direct access to the camera image and various parameters. However, you should used the Player blobfinder proxy, which performs color segmentation and provides a set of bounding boxes around approximately solid color image regions. While laser rangefinders provide more accurate information, cameras are often more practical in terms of their cost, weight, size, and sampling frequency. Cameras have the additional benefit of sensing color, which you will leverage in this assignment.

Given a world with a set of salient objects, your goal for this assignment is to create a Player client that continually drives between these objects. Your controller's decision making should be a finite state machine (FSM). This FSM should use one bit of state to specify the currently sought fiducial. For motion control, you should use proportional-derivative (PD) servoing to center objects in the robot's field of view. As a whole, your controller should put the current object in the center of view, drive as close as possible to an object without hitting it, flip the state bit, and continue the process for the next fiducial.

For object recognition, you will use the blobfinder proxy in Player. The blobfinder proxy examines the images from the camera and groups like-colored pixels of a particular color into regions, or "blobs." The blobfinder proxy returns information about detected blobs and their bounding boxes in the image. The specification for the blobfinder proxy can be found from the Player manual online[1].

3 Assignment Objectives

Your tasks for this assignment are as follows.

1. Generate a color calibration file for the Player blobfinder to recognize the balls (solid yellow) and fiducials.

[1] http://playerstage.sourceforge.net/doc/Player-2.0.0/player/group__player__clientlib__cplusplus.html

2. Write a Player controller to seek out, find, and drive towards a single object.
3. Extend this controller to continually drive between a set of objects in a sequential order.

The following sections provide information to guide you through completing these tasks.

4 Color Calibration

For color blobfinding, Player uses the CMVision library to perform color segmentation and find relatively solid colored regions (or "blobs"), as illustrated in Figure 2. Specifically, Player has a "cmvision" device that uses data that is exposed to the client via the "blobfinder" proxy. The cmvision device processes images provided by a camera device (such as "camera1394") through Player's "camera" proxy. The following is a snippet from a Player configuration file (extension ".cfg", loaded when a Player server is started) reflecting this description:

```
driver
(
  name "cmvision"
  provides ["blobfinder:0"]
  requires ["camera:0"]
  colorfile "blobcolors.txt"
)

driver
(
  name "camera1394"
  provides ["camera:0"]
)
```

The blobfinder specifically provides a bounding box around each image region containg a specified color of interest. To perform blobfinding, Player's cmvision device requires knowledge of these colors of interest through a color calibration file, or "colorfile" ("blobcolors.txt" in the example above). The colorfile contains two sections, listing each blob color in sequential order:

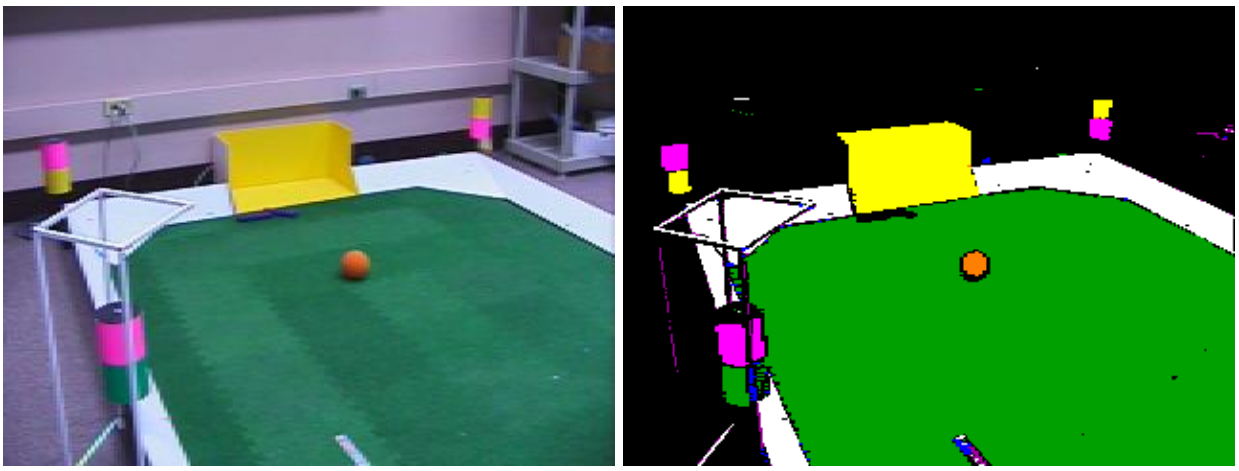

Figure 1: An example of CMVision performing color segmentation on a robot soccer field (from the CMVision website).

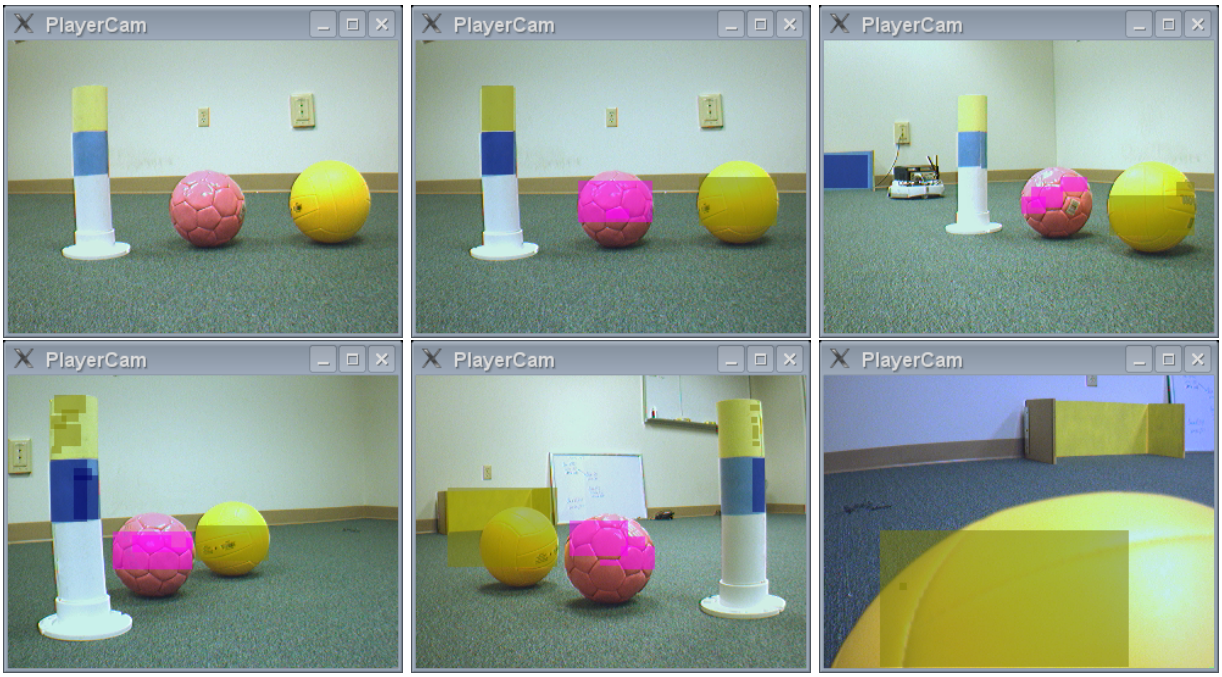

Figure 2: An example of using playercam to color calibrate and perform blobfinding. (Top left) 3 objects are placed in the robot's field of view and playercam is used to extract YUV values and thresholds for each object color. (Other 5 images) Results from blobfinding with the resulting color calibration file from different robot locations and pushing the yellow ball.

- a "[Colors]" section specifying identifiers, as strings and integer triplets, for each blob color
- a "[Thresholds]" section containing color thresholds (in YUV space) associated with each blob color

The following example "blobcolors.txt" illustrates the format of the colorfile:

```
[Colors]
(255,  0,  0) 0.000000 10 Red
(  0,255,  0) 0.000000 10 Green
(  0,  0,255) 0.000000 10 Blue

[Thresholds]
( 25:164, 80:120,150:240)
( 20:220, 50:120, 40:115)
( 15:190,145:255, 40:120)
```

In this colorfile, the color "Red" has the integer identifier "(255,0,0)" or, in hexidecimal, 0x00FF0000 and YUV thresholds (25:164,80:120,150:240). These thresholds are specified as a range. Specifically, any pixel with YUV values within this range will be labelled with as the given blob color. Note: that YUV and RGB color coordinates are vastly different representations, you can refer to the Wikipedia YUV entry and the Appendix for details.

To calibrate the blobfinder, you will need to add appropriate lines in your colorfile containing identifiers and YUV thresholds. Once you have an appropriately calibrated colorfile, the Player blobfinder will be able to detect color blobs, both in real and simulated environments, as illustrated above. We have provided in playercam the ability to output YUV values associated with a pixel currently being displayed. Clicking on a

pixel in playercam will output to the terminal the associated XY location, RGB color values and YUV color values in the following format[2]:

```
[51, 145] = RGB [0 140 0] : : YUV [82 81 69]
```

Given this function in playercam, you can calibrate for the color of specific objects by sampling their pixel colors in playercam. Start playercam and put objects of interest in the robot's view. Click on a set of pixels for a single object that adequately describe the object's color. Note: you may not want to click on all pixels of an object due to shadowing and specular ("shiny") artifacts. From these clicks, playercam will produce a list of YUV values. Examine this list to find appropriate minimum and maximum values for each range. Note: if clicked pixels are chosen carefully, the straightforward min and max in each range can be used. Otherwise, you may want to use some judgment in eliminating noisy outliers.

Once you have a calibrated colorfile, you can restart the Player server and playercam to check the effectiveness of the new thresholds. playercam automatically overlays extracted blobs from the blobfinder using the colorfile in the configuration file. You can use the playerjoy or playerv utilities to then move the robot around the room and perform blobfinding of the objects from different viewpoints. You may find small changes in camera perspective vastly change the performance of the blobfinder in such cases, you can sample pixel color values from these perspectives and adjust your YUV thresholds. Also, make sure to properly order the [Colors] and [Thresholds] sections such that the blob color entries are aligned.

Disclaimer: The calibration process is not always easy and may take several iterations to get a working calibration. Remember, the real world can be particular and unforgiving. Small variations make a huge difference. So, be consistent and thorough.

5 Object Recognition and Seeking

In this assignment, your Player client will need to distinguish three different Roomba Lab objects with the following integer order identifiers[3]:

1. Yellow ball
2. Green over orange fiducial
3. Orange over green fiducial
4. Pink fiducial

Given appropriate color calibration, recognizing single solid color objects should be straightforward. However, fiducials used in robot soccer to indicate specific locations on the field may have multiple solid colors. For example, the playercam image in Figure 3 has two solid colors stacked in a vertical order with similar shape dimensions. In such cases, your Player client will need to specifically include perception routines to process the output of the blobfinder for multicolor fiducials.

Given a specific ordering (via file or command line), your client should drive the robot to visit each of the given Roomba Lab objects continuously in this order. For example, the given ordering [3 1 2 4] should direct the robot to visit the green/orange fiducial, orange/green fiducial, yellow ball, pink fiducial, green/orange fiducial, etc. A finite state machine is a good choice for controlling this decision making. A proportional-derivative feedback controller with a form of wandering is a good choice for motion control.

[2]The binary in */course/cs148/bin/playercam_yuv* is the one on system that provides proper YUV values

[3]Note, these do not necessarily need to be identifiers in the colorfile

6 Teleoperation using playerv

In playerv, the position and bumper proxies from Assignment 1 are still available, but we have also added a blobfinder proxy. Subscribe to the position and blobfinder proxies. The blobfinder proxy should open up a window within playerv displaying the currently detected blobs and their bounding boxes. You may also use the "playercam" application to see the camera images directly, and use playerjoy to control the robot.

After selecting the "command" option for the position proxy, manually navigate the robot (using only the blobfinder for sensing) to the fiducial and ball using playerv. Click and drag the rectangle in the middle of the window to drive the robot. Keep trying manual navigation until you can successfully navigate between the red and blue fiducials using only the blobfinder.

7 Grading

Your grade for Assignment 2 will be determined by equal weighting of your group's implementation (50%) and your individual written report (50%). The weighted breakdown of grading factors for this assignment are as follows:

Project Implementation	
- Color calibration → Is your color calibration file suitable for finding objects in the Roomba Lab? → What are the restrictions (camera pose, lighting, etc.) on your color blobfinding?	20%
- Seeking a single object → Does your robot find, center, drive to non-occluded objects? → How close does your robot get to sought objects?	15%
- Transitioning between objects → Does your robot properly switch to other objects after visiting an object?	8%
- Controller Robustness → Does your controller run without interruption?	7%
Written Report	
- Introduction and Problem Statement → What is your problem? → Why is it interesting?	7%
- Approach and Methods → What is your approach to the problem? → How did you implement your approach and algorithms? → Could someone reproduce your algorithms?	15%
- Experiments and Results → How did you validate your methods? → Describe your variables, controls, and specific tests. → Could someone reproduce your results?	20%
- Conclusion and Discussion → What conclusions can be reached about your problem and approach? → What are the strengths of your approach? → What are the shortcomings of your approach?	8%

Appendix: RGB-YUV Conversions

The color conversion routines used by CMVision for blobfinding in Player are below:

```
#define YUV2RGB(y, u, v, r, g, b)\
  r = y + ((v*1436) >>10);\
  g = y - ((u*352 + v*731) >> 10);\
  b = y + ((u*1814) >> 10);\
  r = r < 0 ? 0 : r;\
  g = g < 0 ? 0 : g;\
  b = b < 0 ? 0 : b;\
  r = r > 255 ? 255 : r;\
  g = g > 255 ? 255 : g;\
  b = b > 255 ? 255 : b

#define RGB2YUV(r, g, b, y, u, v)\
  y = (306*r + 601*g + 117*b)  >> 10;\
  u = ((-172*r - 340*g + 512*b) >> 10)  + 128;\
  v = ((512*r - 429*g - 83*b) >> 10) + 128;\
  y = y < 0 ? 0 : y;\
  u = u < 0 ? 0 : u;\
  v = v < 0 ? 0 : v;\
  y = y > 255 ? 255 : y;\
  u = u > 255 ? 255 : u;\
  v = v > 255 ? 255 : v
```